Project Report

ETS-18

S. N. Landon

A Minicomputer/Microprocessor Programming Interface

19 September 1977

Prepared for the Department of the Air Force
under Electronic Systems Division Contract F19628-76-C-0002 by

Lincoln Laboratory

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

LEXINGTON, MASSACHUSETTS

The work reported in this document was performed at Lincoln Laboratory, a center for research operated by Massachusetts Institute of Technology, with the support of the Department of the Air Force under Contract F19628-76-C-0002.

This technical report has been reviewed and is approved for publication.

FOR THE COMMANDER

Raymond L. Loiselle

Raymond L. Loiselle, Lt. Col., USAF
Chief, ESD Lincoln Laboratory Project Office

MASSACHUSETTS INSTITUTE OF TECHNOLOGY
LINCOLN LABORATORY

A MINICOMPUTER/MICROPROCESSOR PROGRAMMING INTERFACE

S. N. LANDON
Group 94

PROJECT REPORT ETS-18

19 SEPTEMBER 1977

LEXINGTON MASSACHUSETTS

ABSTRACT

A unique solution to the problem of a microcomputer development system is described here. Special-purpose hardware was built for communications between the Motorola 6800 microprocessor and the MODCOMP IV computer. This hardware allows for the transmission of 8 bits of data in either direction at the initiative of the microprocessor. Computer programs were then written for both the microprocessor and the minicomputer to permit the use of the microprocessor Loader and Assembler on source and binary program files stored on the MODCOMP. This system allows the programmer to take advantage of the editing, disc storage and high speed line printer capabilities of the MODCOMP while using the actual M6800 Loader, Assembler and MPU. Sufficiently fast and reliable data transfer rates were achieved such that this system has provided us with an effective development tool. Further, by additions to the programs in the two machines, this interface can easily be expanded to include the M6800 Macro Assembler and Link Loader.

TABLE OF CONTENTS

INTRODUCTION

There are three major approaches to the problem of a microcomputer development system. First, a package may be purchased directly from the manufacturer of the microprocessor; that package interfaces directly to the processor and at best provides a floppy disc, high speed paper tape reader and medium speed line printer. Second, a cross-assembler may be used on a larger machine. This permits faster compilation time and efficient capabilities for file storage and hard copy. Third, a national time-sharing system that provides microcomputer development software may be used.

The first system, that of the microprocessor-based hardware, has the advantage of using the actual microprocessor and its processors. The disadvantages are, however, the cost of the system and the limited speed of the input/output devices connected to it. The second system while faster requires simulation of the microprocessor as well as assurance that the cross-assembler being used is extremely reliable. The third approach has many advantages, particularly if a number of programmers must write programs for a single processor. The continuing cost and the effectiveness of the simulation are, however, factors to assess.

Our application is a small one; we do not require more than a single-user system. We decided to build a hybrid system which combines the features of both of the first two approaches. That system would allow for the storage of data files on the MODCOMP, a minicomputer that we already have. All program editing would be done on that machine. The actual assembling would be done by the microprocessor. The data would be handed to it by the minicomputer when requested by the microprocessor Assembler or Loader. Thus, the actual microprocessor and the interfaced hardware test panel would be used at all stages of program development. Our first attempt at such a system used an RS232C interface to transfer data from the Executive of the M6800 to the MODCOMP. Problems arose due to the lack of hardware handshaking in the link; Motorola's non-standard use of the interface did not promise easy modification. Therefore, we decided to build a special-purpose hardware interface that could transfer data reliably at a high rate - i.e., with handshaking signals. This hardware was relatively simple to build and provided the basis for a more reliable system of communications between the two computers.

The software system was then designed and implemented. As development of a microprocessor-based system of telescope control was underway in parallel with this interface, the system was immediately put to use. An evaluation of this system's performance is therefore also presented here.

I. DESIGN SPECIFICATIONS FOR THE HARDWARE INTERFACE

The specifications for the hardware interface were as follows: all data transfers were to be done at the initiative of the microprocessor. Therefore, whenever the microprocessor performed an input or output operation, the MODCOMP computer was to be interrupted. Until the minicomputer responded and completed the desired operation, a status bit available to the microprocessor would signal that the operation was not yet complete. Upon completion, the bit would change state, signalling the end of the operation.

Eight bits of data were to be transferred because that was a convenient length for the microprocessor. The MODCOMP, however, could only receive and send sixteen bits. The hardware was therefore to handle the stripping off or adding on of the high order byte. Thus, the MODCOMP could send or receive sixteen bits while the microprocessor could send or receive eight.

The final requirement for the hardware was already provided for by the MODCOMP: that the minicomputer have the capability of disabling one or both of the interrupts. This capability simplifies the software design by permitting the data to be single-buffered.

The hardware was built to use two channels in MODCOMP's Input/Output Interface Subsystem (I/OIS).

II. DESIGN OF THE SOFTWARE INTERFACE

A. The Program in the Microprocessor

This system is designed to function at the initiative of the microprocessor. Therefore, communications with the system were planned to occur at the Motorola 6800 console device, which in our case is a teletype. Two commands were implemented. The first invokes the microprocessor Loader and loads the specified binary file from the MODCOMP User Source Library into the memory of the M6800. The second command invokes the M6800 Assembler and assembles a source file from the User Source Library (USL) on the MODCOMP and catalogues the resulting binary code back on USL.

The program in the microprocessor must therefore handle commands from the operator. Those commands are checked for syntax and transmitted unchanged to the MODCOMP computer. The command is also inspected by the microprocessor to determine whether the Loader or the Assembler is to be run. Once the command has been sent to the MODCOMP, the appropriate microprocessor function is begun - assembling or loading.

Special copies of the M6800 Assembler and Loader were created for this system. Those copies differ from the original only in the branch addresses for the input and output routines and in the branch address at the termination of the programs. New input and output routines were written that physically reside within the body of the system interface task.

The input and output routines do the input or output from the MODCOMP via the I/IOS. Status is checked before each I/O operation to ensure that the MODCOMP has finished the last input or output operation. If the status is not good, the routine loops until it is.

With this arrangement, the microprocessor Loader and Assembler function as if they were talking directly to the M6800 console reader, printer and punch. This design allows for simplicity of programming in the microprocessor. Branch addresses must be changed in the processors, and the microprocessor program must handle commands from the operator and provide special-purpose input and output routines.

B. The Software in the Minicomputer

1. The Microprocessor Interface Task

This task handles data that is transmitted to or received from the microprocessor. That data is dealt with in buffers by the input and output handlers, which are discussed below. The transition between records is handled by this program when an entire line has either been transmitted or received. This task suspends its execution while the input and output handlers are processing a record. When one or both of them has finished a record, a bit is set signalling that event. Once this bit has been set, this task wakes up and performs the required functions.

The input and output handlers also set an additional bit to signal what type of operation has just been completed. Each of those functions is then handled by a separate section of this program.

End of Record on Data Input

This end of record condition is the more complex of the two. The data received may be of three types. First, it may be a command line from the microprocessor. If it is not a command, it may either be a line of source listing or of binary code from the Assembler. Each of these three types of data requires different action on the part of this program.

If the data received is a command from the microprocessor, the command must be decoded. Two pieces of data must be obtained from it: whether the operation desired is loading or assembling and the name of the file on USL which is to be transmitted to the microprocessor.

Whether the command is to load or to assemble, a file must be obtained from the User Source Library. This file is obtained by using a special copy of the MODCOMP Source Editor. That copy of the Editor reads commands from a disc partition and can write the desired output onto a second disc partition. When the file is needed, the Microprocessor Interface Task sets up a command file on the appropriate disc partition and starts up the Editor. The Microprocessor Interface Task then monitors the CPU queue. When the Editor has disappeared from the queue, it can safely assume that the copying operation is complete. Thus, the output partition now contains a copy of the file that is to be transmitted to the M6800. That file may be in either Motorola source or binary code (both of which are encoded in ASCII) depending on whether the Assembler or the Loader is to be invoked.

If a Load command is received from the Microprocessor, this task sends the data file to the M6800 only once. For the case of the Assembler, the file is sent twice, once for each pass of the Assembler.

If the data from the microprocessor is determined to be binary code, that data is written onto a special disc partition. The binary code is easily recognized by a header appearing on each line. When the last record of the binary file is received and written onto this partition, the file is catalogued on USL. This cataloguing is done with the special copy of the Editor mentioned above. The command file is set up. That file includes the name under which to catalogue the binary file. That name is the original file name with a "B" appended. The old binary file is removed from USL, and the new one is catalogued in its place.

If the data received from the M6800 is determined to be neither a command nor binary data (note that all of these types may be intermixed), it is then assumed to be a line of the source listing and is routed to the line printer.

Once the data has been processed, the signal bits are reset and the appropriate interrupt is re-enabled. The program will relinquish execution until the input handler signals the next end of record.

End Of Record On Data Output

At the end of record signal on the output operation, the current output record has been completely transmitted and the next one should be placed in the output buffer. This process is handled in the following way: the Source Editor originally wrote the file onto a disc partition. That file is read a record at a time by this task and is put into a buffer for the use of the interrupt handler, which actually transmits the data. Each record has a null at the end of the meaningful data, and transmission stops with the first null encountered or the end of the record (100 bytes), whichever occurs first. At the end of the record, this program is signalled, and it places the next record in the output buffer. When the end of file is reached, the interrupt is not re-enabled as it normally is when the next record is put into the buffer. In any case, the signal bits for the interrupt handlers are reset at the end of the processing.

2. The Input and Output Handlers

The input/output is handled by the addition of two special-purpose interrupt handlers to the MODCOMP Operating System. The initialization of these handlers is further handled by a custom Executive (REX) service, which also functions as a part of the Operating System. That Executive service initializes the dedicated interrupt vector locations and sets up the data link between the interrupt handlers and the Interface Task. This link contains the communications word and the buffer addresses and lengths. The Executive Service also enables the input interrupt so that communications with the microprocessor may begin at any time.

The Input Handler

This handler receives the data from the microprocessor. Before receiving the actual data, the program disables the input interrupt. This prevents the loss of data at the inter-record transition. The microprocessor is capable of generating interrupts faster than the MODCOMP can service them. It is therefore possible that another interrupt could be pending before the handler has determined that this is the end of a record. Consequently, the interrupt is first disabled. It is re-enabled later unless an end of a record has been encountered.

The data is then input from the M6800 and examined. If the character received is a null or a line feed, it is discarded. The character is then stored into the input buffer, which holds a maximum of fifty sixteen bit words. If a carriage return is detected, an end of record has been reached, and further processing must be done. If the character is not a carriage return, the interrupt handler re-enables the input interrupt and exits.

The end of record condition necessitates setting bits for the Interface Task. Also, a null is inserted into the buffer following the last character. Finally, the handler pulses the Taskmaster, the task scheduler in the MODCOMP Operating System. This last function is a requirement placed on interrupt handlers by the MODCOMP I/O System.

Output Handler

The output handler sends data to the microprocessor from the output buffer. Like the input handler, it disables the output interrupt while processing is in progress. This prevents the microprocessor from losing synchronization with the MODCOMP.

The internal buffer pointer is incremented and the appropriate byte is sent to the microprocessor. If the character is a null, it is not transmitted. The null signals the end of the data; in such a case, further processing must be done. Bits must be set for the Interface Task and the MODCOMP Taskmaster pulsed. If this is not the end of the data, the interrupt is re-enabled before the handler exits.

III. FUNCTIONAL CHARACTERISTICS OF THE SYSTEM

The vast majority of the software is invisible to the user of this system. The MODCOMP program is started automatically when the computer system is brought in. Therefore, as long as the MODCOMP system is running, the user need not concern herself with the minicomputer at all. The printout will, of course, appear on the line printer of that machine. Also, the editing of programs must be done on the MODCOMP. However, from the standpoint of using the Loader and the Assembler, the use of the MODCOMP is fully automatic.

All user interactions thus take place at the microprocessor console. The interface program must be started up. The user is prompted with a dot. The command may then be entered -

#LOAD,prognamB

or

#ASM,prognam

The appropriate file will then be loaded or assembled. If the file is not found, error messages appear on the microprocessor console. If for some reason, the microprocessor must be restarted, the ABORT or RESTART key may be pressed. The MODCOMP will receive the new command and cease processing the previous one.

IV. SYSTEM PERFORMANCE EVALUATION

The speed achieved by this interface is quite satisfactory. A program of 2000 lines of assembly code may be assembled in 5-7 minutes. Somewhat more time may elapse while the line printer (600 lines per minute) catches up. The raw transfer times are slower than those achieved by the floppy disc system that Motorola markets. The loading time for the Assembler on our system is 22 seconds compared with Motorola's estimate of 3-4 seconds from the floppy disc. Assembling compares more favorably, however, due to the high speed printer attached to the MODCOMP. As we have not assembled our programs on a floppy disc system, no comparison figures are available. However, even medium speed printers seldom exceed 165 cps; the line printer in use is an order of magnitude faster than that.

One problem has limited the success of this system; that is the difficulties encountered in running this system in a multi-user environment. Our MODCOMP IV has 128K of memory and can support three backgrounds as well as foreground tasks. However, when the microprocessor Interface task is running, certain other operations can crash the machine. This is particularly true of the card reader. The MODCOMP loses track of the priority of the interrupting device and is interrupted as if the requesting device had a priority of zero, which no device has. A halt is planted in the Operating system at that address, and the system dies. As we have seen this problem with other special-purpose handlers written on the MODCOMP, it is not seen as a failing of this particular program. We expect to solve this problem so that this system will be fully compatible with a multi-user system on the MODCOMP. It will remain, however, a single-user system on the M6800.

One of the transitions between commands has also presented some problems. When the hardware for this system becomes available here at the Lab (it is currently available only at our field site), this problem will be fixed.

V. FUTURE PLANS

Based on our success with this approach, we plan to expand the system. Currently, only the Assembler and the absolute code generation capability of the Macro Assembler are used. Future plans include expansion to incorporate the relocatable feature of the Macro Assembler. That change necessitates work in both programs but primarily in the one in the MODCOMP. The data format for the Macro Assembler is different: control characters are interspersed in the data and must be interpreted. Further, this feature requires the Linking Loader, which relocates the code at load time. The Linking Loader has a variety of commands and features including the use of multiple files. Programming must be done - principally in the MODCOMP - to incorporate the Linking Loader.

Greater speed should also be possible. Improvements in the accessing of the records written on disc by using Assembly language and the blocking of records should cut down the disc access time which is currently limiting system performance.

VI. CONCLUSIONS

This system has provided us with a workable microcomputer development system. It is a single user system that provides reasonable response time to the user and the ability to directly program and test within the microprocessor. It has also enabled us to make use of high speed peripheral equipment that we already have - disc packs and a line printer. Use of this system in developing our microprocessor software has proven it to be an effective development tool. Future expansion of the system should make it still more useful. This system is, however, tied very directly to the two particular computers that it was designed for: the MODCOMP IV and the Motorola M6800. If we had plans to use another microprocessor, more development work would have to be done.

ACKNOWLEDGMENTS

I would like to thank L. E. Eaton for the design and checkout of the hardware interface. I am also grateful to P. Wasmund for the microprocessor programming and to A. A. Mathiasen for helpful discussions on the design of the software interface. Thanks are also due to M. A. Grey for her careful typing of this manuscript.

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER ESD-TR-77-232	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) A Minicomputer/Microprocessor Programming Interface		5. TYPE OF REPORT & PERIOD COVERED Project Report
		6. PERFORMING ORG. REPORT NUMBER Project Report ETS-18
7. AUTHOR(s) Susan N. Landon		8. CONTRACT OR GRANT NUMBER(s) F19628-76-C-0002
9. PERFORMING ORGANIZATION NAME AND ADDRESS Lincoln Laboratory, M.I.T. P.O. Box 73 Lexington, MA 02173		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS Program Element No. 63428F Project No. 2128
11. CONTROLLING OFFICE NAME AND ADDRESS Air Force Systems Command, USAF Andrews AFB Washington, DC 20331		12. REPORT DATE 19 September 1977
		13. NUMBER OF PAGES 22
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office) Electronic Systems Division Hanscom AFB Bedford, MA 01731		15. SECURITY CLASS. (of this report) Unclassified
		15a. DECLASSIFICATION DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

None

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

GEODSS
programming
minicomputers
microprocessors

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

A unique solution to the problem of a minicomputer development system is described here. Special-purpose hardware was built for communications between the Motorola 6800 microprocessor and the MODCOMP IV computer. This hardware allows for the transmission of 8 bits of data in either direction at the initiative of the microprocessor. Computer programs were then written for both the microprocessor and the minicomputer to permit the use of the microprocessor Loader and Assembler on source and binary program files stored on the MODCOMP. This system allows the programmer to take advantage of the editing, disc storage and high speed line printer capabilities of the MODCOMP while using the actual M6800 Loader, Assembler and MPU. Sufficiently fast and reliable data transfer rates were achieved such that this system has provided us with an effective development tool. Further, by additions to the programs in the two machines, this interface can easily be expanded to include the M6800 Macro Assembler and Link Loader.

DD FORM 1473 1 JAN 73 EDITION OF 1 NOV 65 IS OBSOLETE

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)